how to build a twitter sentiment analyzer ? (.)

Ravikiran Janardhana — May 08, 2012, 05:27 PM (.) — 186 Comments (.#disqus_thread)

UPDATE: The github repo for twitter sentiment analyzer (https://github.com/ravikiranj/twitter-sentiment-analyzer) now contains updated get_twitter_data.py (https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/get_twitter_data.py) file compatible with Twitter API v1.1. It can be tested by placing appropriate oauth credentials in config.json (https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/config.json) and running test_twitter_data.py (https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/test_twitter_data.py). You can create a new twitter app at https://dev.twitter.com/apps (https://dev.twitter.com/apps) to fetch necessary oauth credentials.

Hi all, It's been almost a year since I last wrote a technical post. A lot of changes have occurred in my life since then, from a Frontend engineer at Yahoo!, I've transformed into a full-time graduate student at UNC-Chapel Hill who is moving to Redmond to do an internship at Microsoft this summer. In my spring semester, I took Data Mining course for which I had to complete a project as part of the course. After exploring various ideas, I finalized on building a Twitter Sentiment Analyzer. This project aimed to extract tweets about a particular topic from twitter (recency = 1-7 days) and analyze the opinion of tweeples (people who use twitter.com) on this topic as positive, negative or neutral. In this post, I will explain you how you can build such a sentiment analyzer. I will try to explain the concepts without making it sound too technical, but a good knowledge of machine learning classifiers really helps.

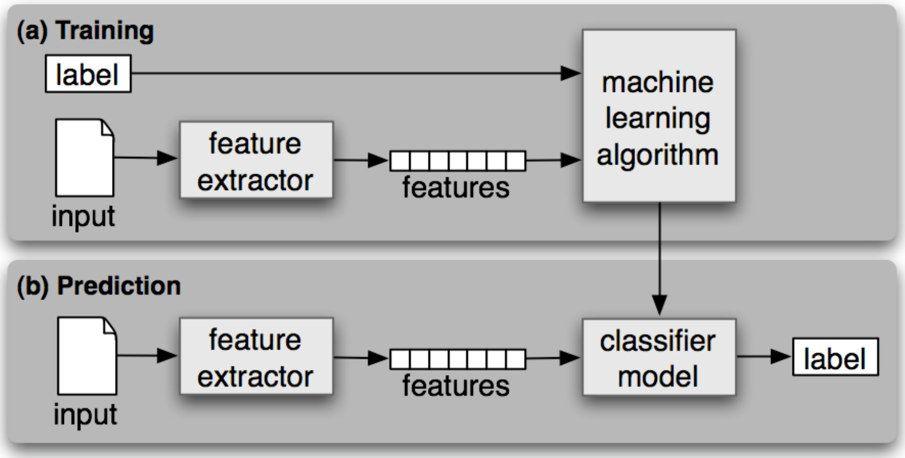
Motivation

Twitter is a popular micro blogging service where users create status messages (called “tweets”). These tweets sometimes express opinions about different topics. I propose to build an automatic sentiment (positive or neutral or negative) extractor from a tweet. This is very useful because it allows feedback to be aggregated without manual intervention. Using this analyzer,

- Consumers can use sentiment analysis to research products or services before making a purchase. E.g. Kindle
- Marketers can use this to research public opinion of their company and products, or to analyze customer satisfaction. E.g. Election Polls
- Organizations can also use this to gather critical feedback about problems in newly released products. E.g. Brand Management (Nike, Adidas)

Background

In order to build a sentiment analyzer, first we need to equip ourselves with the right tools and methods. Machine learning is one such tool where people have developed various methods to classify. Classifiers may or may not need training data. In particular, we will deal with the following machine learning classifiers, namely, Naive Bayes Classifier, Maximum Entropy Classifier and Support Vector Machines. All of these classifiers require training data and hence these methods fall under the category of supervised classification.



Supervised Classification (Original Source (http://www.nltk.org/book/ch06.html))

To get a good understanding of how these algorithms work, I would suggest you to refer any of the standard machine learning / data mining books. To get a nice overview, you can refer to B. Pang and L. Lee. Opinion mining and sentiment analysis (http://www.cs.cornell.edu/home/llee/omsa/omsa.pdf).

Implementation Details

I will be using Python (2.x or 3.x) along with the Natural Language Toolkit (nltk) and libsvm libraries to implement the classifiers. You can use webpy library if you want to build a web interface. If you are using Ubuntu, you can get all of these with a single command as below.

```
sudo apt-get install python python-nltk python-libsvm python-yaml python-webpy python-oauth2
```

Training the Classifiers

The classifiers need to be trained and to do that, we need to list manually classified tweets. Let's start with 3 positive, 3 neutral and 3 negative tweets.

Positive tweets

1. @PrincessSuperC Hey Cici sweetheart! Just wanted to let u know I luv u! OH! and will the mixtape drop soon? FANTASY RIDE MAY 5TH!!!!
2. @Msdebramaye I heard about that contest! Congrats girl!!
3. UNC!!! NCAA Champs!! Franklin St.: I WAS THERE!! WILD AND CRAZY!!!!!! Nothing like it...EVER http://tinyurl.com/49955t3 (http://tinyurl.com/49955t3)

Neutral tweets

1. Do you Share More #jokes #quotes #music #photos or #news #articles on #Facebook or #Twitter?
2. Good night #Twitter and #TheLegionoftheFallen. 5:45am cimes awfully early!
3. I just finished a 2.66 mi run with a pace of 11'14"/mi with Nike+ GPS. #nikeplus #makeitcount

Negative tweets

1. Disappointing day. Attended a car boot sale to raise some funds for the sanctuary, made a total of 88p after the entry fee - sigh

2. no more taking Irish car bombs with strange Australian women who can drink like rockstars...my head hurts.
3. Just had some bloodwork done. My arm hurts

As you can see from above, the tweets can have some valuable info about it's sentiment and rest of the words may not really help in determining the sentiment. Therefore, it makes sense to preprocess the tweets.

Preprocess tweets

1. Lower Case - Convert the tweets to lower case.
2. URLs - I don't intend to follow the short urls and determine the content of the site, so we can eliminate all of these URLs via regular expression matching or replace with generic word URL.
3. @username - we can eliminate "@username" via regex matching or replace it with generic word AT_USER.
4. #hashtag - hash tags can give us some useful information, so it is useful to replace them with the exact same word without the hash. E.g. #nike replaced with 'nike'.
5. Punctuations and additional white spaces - remove punctuation at the start and ending of the tweets. E.g: ' the day is beautiful! ' replaced with 'the day is beautiful'. It is also helpful to replace multiple whitespaces with a single whitespace

Code - Preprocess tweets

```
#import regex
import re

#start process_tweet
def processTweet(tweet):
    # process the tweets

    #Convert to lower case
    tweet = tweet.lower()
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
    #Convert @username to AT_USER
    tweet = re.sub('@[^\s]+','AT_USER',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Replace #word with word
    tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
    #trim
    tweet = tweet.strip('\'"')
    return tweet
#end

#Read the tweets one by one and process it
fp = open('data/sampleTweets.txt', 'r')
line = fp.readline()

while line:
    processedTweet = processTweet(line)
    print processedTweet
    line = fp.readline()
#end loop
fp.close()
```

After processing, the same tweets look as below.

Positive tweets

1. AT_USER hey cici sweetheart! just wanted to let u know i luv u! oh! and will the mixtape drop soon? fantasy ride may 5th!!!!
2. AT_USER i heard about that contest! congrats girl!!
3. unc!!! ncaa champs!! franklin st.: i was there!! wild and crazy!!!!!! nothing like it...ever URL

Neutral tweets

1. do you share more jokes quotes music photos or news articles on facebook or twitter?
2. good night twitter and thelegionofthefallen. 5:45am cimes awfully early!
3. i just finished a 2.66 mi run with a pace of 11:14/mi with nike+ gps. nikeplus makeitcount

Negative tweets

1. disappointing day. attended a car boot sale to raise some funds for the sanctuary, made a total of 88p after the entry fee - sigh
2. no more taking irish car bombs with strange australian women who can drink like rockstars...my head hurts.
3. just had some bloodwork done. my arm hurts

Feature Vector

Feature vector is the most important concept in implementing a classifier. A good feature vector directly determines how successful your classifier will be. The feature vector is used to build a model which the classifier learns from the training data and further can be used to classify previously unseen data.

To explain this, I will take a simple example of "gender identification". Male and Female names have some distinctive characteristics. Names ending in a, e and i are likely to be female, while names ending in k, o, r, s and t are likely to be male. So, you can build a classifier based on this model using the ending letter of the names as a feature.

Similarly, in tweets, we can use the presence/absence of words that appear in tweet as features. In the training data, consisting of positive, negative and neutral tweets, we can split each tweet into words and add each word to the feature vector. Some of the words might not have any say in indicating the sentiment of a tweet and hence we can filter them out. Adding individual (single) words to the feature vector is referred to as 'unigrams' approach.

Some of the other feature vectors also add 'bi-grams' in combination with 'unigrams'. For example, 'not good' (bigram) completely changes the sentiment compared to adding 'not' and 'good' individually. Here, for simplicity, we will only consider the unigrams. Before adding the words to the feature vector, we need to preprocess them in order to filter, otherwise, the feature vector will explode.

Filtering tweet words (for feature vector)

1. Stop words - a, is, the, with etc. The full list of stop words can be found at Stop Word List (https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/data/feature_list/stopwords.txt). These words don't indicate any sentiment and can be removed.
2. Repeating letters - if you look at the tweets, sometimes people repeat letters to stress the emotion. E.g. hunggrryyy, huuuuuuungry for 'hungry'. We can look for 2 or more repetitive letters in words and replace them by 2 of the same.
3. Punctuation - we can remove punctuation such as comma, single/double quote, question marks at the start and end of each word. E.g. beautiful!!!!!! replaced with beautiful
4. Words must start with an alphabet - For simplicity sake, we can remove all those words which don't start with an alphabet. E.g. 15th, 5.34am

Code - Filtering tweet words (for feature vector)

```
#initialize stopWords
stopWords = []

#start replaceTwoOrMore
def replaceTwoOrMore(s):
    #look for 2 or more repetitions of character and replace with the character itself
    pattern = re.compile(r"(.)\1{1,}", re.DOTALL)
    return pattern.sub(r"\1\1", s)
#end

#start getStopWordList
```

```
def getStopWordList(stopWordListFileName):
    #read the stopwords file and build a list
    stopWords = []
    stopWords.append('AT_USER')
    stopWords.append('URL')

    fp = open(stopWordListFileName, 'r')
    line = fp.readline()
    while line:
        word = line.strip()
        stopWords.append(word)
        line = fp.readline()
    fp.close()
    return stopWords
#end

#start getfeatureVector
def getFeatureVector(tweet):
    featureVector = []
    #split tweet into words
    words = tweet.split()
    for w in words:
        #replace two or more with two occurrences
        w = replaceTwoOrMore(w)
        #strip punctuation
        w = w.strip('\'"?,.')
        #check if the word stats with an alphabet
        val = re.search(r"^[a-zA-Z][a-zA-Z0-9]*$", w)
        #ignore if it is a stop word
        if(w in stopWords or val is None):
            continue
        else:
            featureVector.append(w.lower())
    return featureVector
#end

#Read the tweets one by one and process it
fp = open('data/sampleTweets.txt', 'r')
line = fp.readline()

st = open('data/feature_list/stopwords.txt', 'r')
stopWords = getStopWordList('data/feature_list/stopwords.txt')

while line:
    processedTweet = processTweet(line)
    featureVector = getFeatureVector(processedTweet)
    print featureVector
    line = fp.readline()
```

```
#end loop
fp.close()
```

As we process, each of the tweets, we keep adding words to the feature vector and ignoring other words. Let us look at the feature words extracted for the tweets.

Positive Tweets	Feature Words
AT_USER hey cici sweetheart! just wanted to let u know i luv u! oh! and will the mixtape drop soon? fantasy ride may 5th!!!!	'hey', 'cici', 'luv', 'mixtape', 'drop', 'soon', 'fantasy', 'ride'
AT_USER i heard about that contest! congrats girl!!	'heard', 'congrats'
unc!!! ncaa champs!! franklin st.: i was there!! wild and crazy!!!!!! nothing like it...ever URL	'ncaa', 'franklin', 'wild'

Neutral Tweets	Feature Words
do you share more jokes quotes music photos or news articles on facebook or twitter?	'share', 'jokes', 'quotes', 'music', 'photos', 'news', 'articles', 'facebook', 'twitter'
good night twitter and thelegionofthefallen. 5:45am cimes awfully early!	'night', 'twitter', 'thelegionofthefallen', 'cimes', 'awfully'
i just finished a 2.66 mi run with a pace of 11:14/mi with nike+ gps. nikeplus makeitcount	'finished', 'mi', 'run', 'pace', 'gps', 'nikeplus', 'makeitcount'

Negative Tweets	Feature Words
disappointing day. attended a car boot sale to raise some funds for the sanctuary, made a total of 88p after the entry fee - sigh	'disappointing', 'day', 'attended', 'car', 'boot', 'sale', 'raise', 'funds', 'sanctuary', 'total', 'entry', 'fee', 'sigh'
no more taking irish car bombs with strange australian women who can drink like rockstars...my head hurts.	'taking', 'irish', 'car', 'bombs', 'strange', 'australian', 'women', 'drink', 'head', 'hurts'
just had some bloodwork done. my arm hurts	'bloodwork', 'arm', 'hurts'

The entire feature vector will be a combination of each of these feature words. For each tweet, if a feature word is present, we mark it as 1, else marked as 0. Instead of using presence/absence of feature word, you may also use the count of it, but since tweets are just 140 chars, I use 0/1. Now, you can think of each tweet as a bunch of 1s and 0s and based on this pattern, a tweet is labeled as positive, neutral or negative.

Given any new tweet, we need to extract the feature words as above and we get one more pattern of 0s and 1s and based on the model learned, the classifiers predict the tweet sentiment. It's highly essential for you to understand this point and I have to tried to make it as simple as possible. If you don't get how the sentiment is extracted, go re-read from the top or refer a good machine learning / data mining book on classifiers.

In my full implementation, I used the method of distant supervision to obtain a large training dataset. This method is detailed out in Twitter Sentiment Classification using Distant Supervision (http://www.citeulike.org/user/pcalado/article/8047905). For the following sections, I assume that you have a list of large training dataset in CSV or some other format, which you can load and train the classifiers. You can look at below webpages for training datasets.

- Twitter sentiment dataset - 2008 US Election debate by Nick Diakopoulos and Shamma, D.A. (http://www.ayman-naaman.net/2010/11/21/twitter-sentiment-dataset-online/)

- Twitter sentiment corpus by Niek Sanders (http://www.sananalytics.com/lab/twitter-sentiment/)
- A lot of sentiment datasets via CS Dept, Cornell University (http://www.cs.cornell.edu/people/pabo/movie-review-data/)

Let's get the ball rolling

Now that I have covered enough of background information on classifiers, now its time to take a look at Natural Language Toolkit (NLTK) and implement the first two classifiers namely Naive Bayes and Maximum Entropy.

For the explanation, I will use a sample CSV file consisting of labeled tweets, the contents of the CSV file are as below.

sampleTweets.csv

```
|positive|,|@PrincessSuperC Hey Cici sweetheart! Just wanted to let u know I luv u! OH! and will the mixta
|positive|,|@Msdebramaye I heard about that contest! Congrats girl!!|
|positive|,|UNC!!! NCAA Champs!! Franklin St.: I WAS THERE!! WILD AND CRAZY!!!!!! Nothing like it...EVER h
|neutral|,|Do you Share More #jokes #quotes #music #photos or #news #articles on #Facebook or #Twitter?|
|neutral|,|Good night #Twitter and #TheLegionoftheFallen.  5:45am cimes awfully early!|
|neutral|,|I just finished a 2.66 mi run with a pace of 11'14"/mi with Nike+ GPS. #nikeplus #makeitcount|
|negative|,|Disappointing day. Attended a car boot sale to raise some funds for the sanctuary, made a tota
|negative|,|no more taking Irish car bombs with strange Australian women who can drink like rockstars...my
|negative|,|Just had some bloodwork done. My arm hurts|
```

The following code, extracts the tweets and label from the csv file and processes it as outlined above and obtains a feature vector and stores it in a variable called "tweets".

Feature Extraction

```python
#Read the tweets one by one and process it
inpTweets = csv.reader(open('data/sampleTweets.csv', 'rb'), delimiter=',', quotechar='|')
tweets = []
for row in inpTweets:
    sentiment = row[0]
    tweet = row[1]
    processedTweet = processTweet(tweet)
    featureVector = getFeatureVector(processedTweet, stopWords)
    tweets.append((featureVector, sentiment));
#end loop
```

Tweets Variable

```
tweets = [(['hey', 'cici', 'luv', 'mixtape', 'drop', 'soon', 'fantasy', 'ride'], 'positive'),
            (['heard', 'congrats'], 'positive'),
            (['ncaa', 'franklin', 'wild'], 'positive'),
            (['share', 'jokes', 'quotes', 'music', 'photos', 'news', 'articles', 'facebook', 'twitter'],
            (['night', 'twitter', 'thelegionofthefallen', 'cimes', 'awfully'], 'neutral'),
            (['finished', 'mi', 'run', 'pace', 'gps', 'nikeplus', 'makeitcount'], 'neutral'),
            (['disappointing', 'day', 'attended', 'car', 'boot', 'sale', 'raise', 'funds', 'sanctuary',
              'total', 'entry', 'fee', 'sigh'], 'negative'),
            (['taking', 'irish', 'car', 'bombs', 'strange', 'australian', 'women', 'drink', 'head',
              'hurts'], 'negative'),
            (['bloodwork', 'arm', 'hurts'], 'negative')]
```

Our big feature vector now consists of all the feature words extracted from tweets. Let us call this "featureList", now we need to write a method, which gives us the crisp feature vector for all tweets, which we can use to train the classifier.

Feature List

```
featureList = ['hey', 'cici', 'luv', 'mixtape', 'drop', 'soon', 'fantasy', 'ride', 'heard',
'congrats', 'ncaa', 'franklin', 'wild', 'share', 'jokes', 'quotes', 'music', 'photos', 'news',
'articles', 'facebook', 'twitter', 'night', 'twitter', 'thelegionofthefallen', 'cimes', 'awfully',
'finished', 'mi', 'run', 'pace', 'gps', 'nikeplus', 'makeitcount', 'disappointing', 'day', 'attended',
'car', 'boot', 'sale', 'raise', 'funds', 'sanctuary', 'total', 'entry', 'fee', 'sigh', 'taking',
'irish', 'car', 'bombs', 'strange', 'australian', 'women', 'drink', 'head', 'hurts', 'bloodwork',
'arm', 'hurts']
```

Extract Features Method

```
#get feature list stored in a file (for reuse)
featureList = getFeatureList('data/sampleTweetFeatureList.txt')

#start extract_features
def extract_features(tweet):
    tweet_words = set(tweet)
    features = {}
    for word in featureList:
        features['contains(%s)' % word] = (word in tweet_words)
    return features
#end
```

Output of Extract Features

Consider a sample tweet *"just had some bloodwork done. my arm hurts"*, the feature words extracted for this tweet is *['bloodwork', 'arm', 'hurts']*. If we pass this list as an input to *extract_features* method which makes use of the *'featureList'* , the output obtained is as below.

```
{
    'contains(arm)': True,             #notice this
    'contains(articles)': False,
    'contains(attended)': False,
    'contains(australian)': False,
    'contains(awfully)': False,
    'contains(bloodwork)': True,       #notice this
    'contains(bombs)': False,
    'contains(cici)': False,
    .....
    'contains(head)': False,
    'contains(heard)': False,
    'contains(hey)': False,
    'contains(hurts)': True,           #notice this
    .....
    'contains(irish)': False,
    'contains(jokes)': False,
    .....
    'contains(women)': False
}
```

Bulk Extraction of Features

NLTK has a neat feature which enables to extract features as above in bulk for all the tweets and can be done using the below code snippet. The line of interest is *"nltk.classify.apply_features(extract_features, tweets)"* where you pass in the tweets variable to the *extract_features* method.

```
#Read the tweets one by one and process it
inpTweets = csv.reader(open('data/sampleTweets.csv', 'rb'), delimiter=',', quotechar='|')
stopWords = getStopWordList('data/feature_list/stopwords.txt')
featureList = []

# Get tweet words
tweets = []
for row in inpTweets:
    sentiment = row[0]
    tweet = row[1]
    processedTweet = processTweet(tweet)
    featureVector = getFeatureVector(processedTweet, stopWords)
    featureList.extend(featureVector)
    tweets.append((featureVector, sentiment));
#end loop

# Remove featureList duplicates
featureList = list(set(featureList))

# Extract feature vector for all tweets in one shote
training_set = nltk.classify.util.apply_features(extract_features, tweets)
```

Both the Naive Bayes and Maximum Entropy Classifier have exactly the same steps until this point and vary slightly from here on.

Naive Bayes Classifier

To explain how a Naive Bayes Classifier works is beyond the scope of this post, having said so, its pretty easy to understand. Refer to the Wikipedia article (http://en.wikipedia.org/wiki/Naive_Bayes_classifier) and read the example to understand how it works. At this point, we have a training set, so all we need to do is instantiate a classifier and classify test tweets. The below code explains how to classify a single tweet using the classifier.

```
# Train the classifier
NBClassifier = nltk.NaiveBayesClassifier.train(training_set)

# Test the classifier
testTweet = 'Congrats @ravikiranj, i heard you wrote a new tech post on sentiment analysis'
processedTestTweet = processTweet(testTweet)
print NBClassifier.classify(extract_features(getFeatureVector(processedTestTweet)))

#Output
#======
#positive
```

Informative Features

NLTK has a neat feature of printing out the most informative features using the below piece of code.

```
# print informative features about the classifier
print NBClassifier.show_most_informative_features(10)

# Output
# ======
# Most Informative Features
#     contains(twitter) = False          positi : neutra  =      2.3 : 1.0
#         contains(car) = False          positi : negati  =      2.3 : 1.0
#       contains(hurts) = False          positi : negati  =      2.3 : 1.0
#    contains(articles) = False          positi : neutra  =      1.4 : 1.0
#       contains(heard) = False          neutra : positi  =      1.4 : 1.0
#         contains(hey) = False          neutra : positi  =      1.4 : 1.0
#       contains(total) = False          positi : negati  =      1.4 : 1.0
#          contains(mi) = False          positi : neutra  =      1.4 : 1.0
#         contains(day) = False          positi : negati  =      1.4 : 1.0
#contains(makeitcount) = False          positi : neutra  =      1.4 : 1.0
```

I highly recommend you to lookup Laurent Luce's brilliant post on digging up the internals of nltk classifier at Twitter Sentiment Analysis using Python and NLTK (http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/). If you do have a test set of manually labeled data, you can cross verify it via the classifier. You will soon find that the results are not so good as you expected (see below).

```
testTweet = 'I am so badly hurt'
processedTestTweet = processTweet(testTweet)
print NBClassifier.classify(extract_features(getFeatureVector(processedTestTweet)))

#Output
#======
#positive
```

This is essentially because in the training data didn't cover the words encountered in this tweet and the classifier has little knowledge to classify this tweet and most often the tweet gets assigned the default classification label, in this case happens to be 'positive'. Hence, training dataset is very crucial for the success of these classifiers. Anything below 10k of training tweets will give you pretty mediocre results.

Maximum Entropy Classifier

The explanation of how a maximum entropy classifier works is beyond the scope of this post. You can refer Using Maximum Entropy for Text Classification (http://www.kamalnigam.com/papers/maxent-ijcaiws99.pdf) to get a good idea of how it works. There are a lot of options available when instantiating the Maximum Entropy classifier, all of which are explained at NLTK Maxent Classifier Class documentation (http://www.nltk.org/api/nltk.classify.html#module-nltk.classify.maxent). I use the 'General Iterative Scaling' algorithm and usually stick to 10 iterations. You can also extract the most informative features as before which gives a good idea of how the classifier works. The code to instantiate the classifier and to classify tweets is as below.

```
#Max Entropy Classifier
MaxEntClassifier = nltk.classify.maxent.MaxentClassifier.train(training_set, 'GIS', trace=3, \
                    encoding=None, labels=None, sparse=True, gaussian_prior_sigma=0, max_iter = 10)
testTweet = 'Congrats @ravikiranj, i heard you wrote a new tech post on sentiment analysis'
processedTestTweet = processTweet(testTweet)
print MaxEntClassifier.classify(extract_features(getFeatureVector(processedTestTweet)))

# Output
# =======
# positive

#print informative features
print MaxEntClassifier.show_most_informative_features(10)

# Output
# =======
# ==> Training (10 iterations)
#
#      Iteration    Log Likelihood    Accuracy
#      ---------------------------------------
#             1          -1.09861        0.333
#             2          -0.86350        1.000
#             3          -0.69357        1.000
#             4          -0.57184        1.000
#             5          -0.48323        1.000
#             6          -0.41705        1.000
#             7          -0.36625        1.000
#             8          -0.32624        1.000
#             9          -0.29401        1.000
#         Final          -0.26751        1.000
#  -0.269 Correction feature (58)
#   0.192 contains(arm)==True and label is 'negative'
#   0.192 contains(bloodwork)==True and label is 'negative'
#   0.168 contains(congrats)==True and label is 'positive'
#   0.168 contains(heard)==True and label is 'positive'
#   0.152 contains(franklin)==True and label is 'positive'
#   0.152 contains(wild)==True and label is 'positive'
#   0.152 contains(ncaa)==True and label is 'positive'
#   0.147 contains(night)==True and label is 'neutral'
#   0.147 contains(awfully)==True and label is 'neutral'
```

Support Vector Machines

Support Vector Machines (SVM) is pretty much the standard classifier which is used for any general purpose classification. As the earlier methods, explaining how SVM works will itself take an entire post. Please refer to the Wikipedia article on SVM (http://en.wikipedia.org/wiki/Support_vector_machine) to understand how it works. I will use the libsvm library (http://www.csie.ntu.edu.tw/~cjlin/libsvm/) (written in C++ and has a python handle) implemented by Chih-Chung Chang and Chih-Jen Lin to instantiate SVM. Detailed documentation of the python handle can be read in the libsvm.tar.gz (http://www.csie.ntu.edu.tw/~cjlin/cgi-bin/libsvm.cgi?+http://www.csie.ntu.edu.tw/~cjlin/libsvm+tar.gz) extracted folder or libsvm github repo (https://github.com/cjlin1/libsvm).

To make you understand how it works, I will first implement a simple example of classification and extend the same idea to the tweet sentiment classifier.

Consider that you have 3 set of labels (0, 1, 2) and series of 0s and 1s indicate what label they belong too. We will train a *LINEAR SVM* classifier based on this training data. I will also show how you can save this model for future reuse so that you don't need to train them again. The test data will also comprise of a series of 0s and 1s and now we need to predict the label from the label set = {0, 1, 2}. The below example exactly does what's described in this paragraph.

```
import svm
from svmutil import *

#training data
labels = [0, 1, 1, 2]
samples = [[0, 1, 0], [1, 1, 1], [1, 1, 0], [0, 0, 0]]

#SVM params
param = svm_parameter()
param.C = 10
param.kernel_type = LINEAR
#instantiate the problem
problem = svm_problem(labels, samples)
#train the model
model = svm_train(problem, param)
# saved model can be loaded as below
#model = svm_load_model('model_file')

#save the model
svm_save_model('model_file', model)

#test data
test_data = [[0, 1, 1], [1, 0, 1]]
#predict the labels
p_labels, p_accs, p_vals = svm_predict([0]*len(test_data), test_data, model)
print p_labels
```

The output of the above code is as below:-

```
.*
optimization finished, #iter = 5
nu = 0.176245
obj = -2.643822, rho = 0.164343
nSV = 3, nBSV = 0
*
optimization finished, #iter = 1
nu = 0.254149
obj = -2.541494, rho = 0.000000
nSV = 2, nBSV = 0
.*.*
optimization finished, #iter = 6
nu = 0.112431
obj = -1.686866, rho = -0.143522
nSV = 3, nBSV = 0
Total nSV = 4
Accuracy = 50% (1/2) (classification)
[0.0, 1.0]
```

For the test data, the predicted label was 0 for the first case and 1 for the second case. Internally, the classifier does a cross validation on the training data and outputs the accuracy as 50% which indicates that we need to add more test data to improve the accuracy (given that the results are not totally random!).

Now consider our *sampleTweets.csv* file, the featureList vector will be as shown below and when we process a sentence, the column values of the unigram features will be set to 1, other wise 0. A combination of these 0s and 1s in the feature vector along with the known label will be the training input to our SVM classifier. It should be noted that the label in the feature vector should be numeric only for the SVM classifier. Hence, I use 0 for positive, 1 for negative and 2 for neutral labels.

```
Sentence = AT_USER i heard about that contest! congrats girl!!

Feature Vector
==============
hey',.....'heard','congrats', .... 'bombs', 'strange', 'australian', 'women', 'drink', 'head', 'hurts', 'b
 0           1        1               0        0            0           0        0       0        0
```

The following code shows how to extract the feature vector for the SVM classifier and also the code sample to train and test the SVM classifier.

```
def getSVMFeatureVectorAndLabels(tweets, featureList):
    sortedFeatures = sorted(featureList)
    map = {}
    feature_vector = []
    labels = []
    for t in tweets:
        label = 0
        map = {}
        #Initialize empty map
        for w in sortedFeatures:
            map[w] = 0

        tweet_words = t[0]
        tweet_opinion = t[1]
        #Fill the map
        for word in tweet_words:
            #process the word (remove repetitions and punctuations)
            word = replaceTwoOrMore(word)
            word = word.strip('\'"?,.')
            #set map[word] to 1 if word exists
            if word in map:
                map[word] = 1
        #end for loop
        values = map.values()
        feature_vector.append(values)
        if(tweet_opinion == 'positive'):
            label = 0
        elif(tweet_opinion == 'negative'):
            label = 1
        elif(tweet_opinion == 'neutral'):
            label = 2
        labels.append(label)
    #return the list of feature_vector and labels
    return {'feature_vector' : feature_vector, 'labels': labels}
#end

#Train the classifier
result = getSVMFeatureVectorandLabels(tweets, featureList)
problem = svm_problem(result['labels'], result['feature_vector'])
#'-q' option suppress console output
param = svm_parameter('-q')
param.kernel_type = LINEAR
classifier = svm_train(problem, param)
svm_save_model(classifierDumpFile, classifier)

#Test the classifier
test_feature_vector = getSVMFeatureVector(test_tweets, featureList)
#p_labels contains the final labeling result
p_labels, p_accs, p_vals = svm_predict([0] * len(test_feature_vector),test_feature_vector, classifier)
```

Phew, I hope you are now armed with the ability to classify the sentiment of any sentence using the above mentioned classifiers. Now, let us talk something about twitter :) !

Retrieving tweets for a particular topic

When you build a twitter sentiment analyzer, the input to your system will be a user enter keyword. Hence, one of the building blocks of this system will be to fetch tweets based on the keyword within a selected time duration.

The most important reference to achieve this is the Twitter API Documentation for Tweet Search (https://dev.twitter.com/rest/reference/get/search/tweets). There are a lot of options that you can set in the API query and for the purpose of demonstrating the API, I will use only the simpler options.

The API endpoint I would hit for purpose of demonstration is as below.

```
https://api.twitter.com/1.1/search/tweets.json?q=keyword&lang=en&result_type=recent&count=100&include_enti
```

The following code shows how you can retrieve the tweets given a particular keyword. You need to specify config.json (https://github.com/ravikiranj/twitter-sentiment-analyzer/blob/master/config.json) as defined below so that oauth requests can be made.

config.json

```
{
    "consumer_key": "YOUR CONSUMER KEY",
    "consumer_secret": "YOUR CONSUMER SECRET",
    "access_token": "YOUR ACCESS TOKEN",
    "access_token_secret": "YOUR ACCESS TOKEN SECRET"
}
```

get_twitter_data.py

```
import argparse
import urllib
import json
import os
import oauth2

class TwitterData:
    def parse_config(self):
        config = {}
        # from file args
        if os.path.exists('config.json'):
            with open('config.json') as f:
                config.update(json.load(f))
        else:
            # may be from command line
            parser = argparse.ArgumentParser()

            parser.add_argument('-ck', '--consumer_key', default=None, help='Your developper `Consumer K
            parser.add_argument('-cs', '--consumer_secret', default=None, help='Your developper `Consume
            parser.add_argument('-at', '--access_token', default=None, help='A client `Access Token`')
            parser.add_argument('-ats', '--access_token_secret', default=None, help='A client `Access To

            args_ = parser.parse_args()
            def val(key):
                return config.get(key)\
                    or getattr(args_, key)\
                    or raw_input('Your developper `%s`: ' % key)
            config.update({
                'consumer_key': val('consumer_key'),
                'consumer_secret': val('consumer_secret'),
                'access_token': val('access_token'),
                'access_token_secret': val('access_token_secret'),
            })
        # should have something now
        return config
    #end

    def oauth_req(self, url, http_method="GET", post_body=None,
                  http_headers=None):
        config = self.parse_config()
        consumer = oauth2.Consumer(key=config.get('consumer_key'), secret=config.get('consumer_secret'))
        token = oauth2.Token(key=config.get('access_token'), secret=config.get('access_token_secret'))
        client = oauth2.Client(consumer, token)

        resp, content = client.request(
            url,
            method=http_method,
            body=post_body or '',
            headers=http_headers
        )
        return content
    #end

    #start getTwitterData
    def getData(self, keyword, params = {}):
        maxTweets = 50
        url = 'https://api.twitter.com/1.1/search/tweets.json?'
        data = {'q': keyword, 'lang': 'en', 'result_type': 'recent', 'count': maxTweets, 'include_entiti

        #Add if additional params are passed
        if params:
            for key, value in params.iteritems():
                data[key] = value

        url += urllib.urlencode(data)

        response = self.oauth_req(url)
        jsonData = json.loads(response)
        tweets = []
        if 'errors' in jsonData:
            print "API Error"
            print jsonData['errors']
        else:
            for item in jsonData['statuses']:
                tweets.append(item['text'])
        return tweets
    #end
#end class
```

```

## Usage
## =====
## td = TwitterData()
## print td.getData('barca')
```

Putting it all together

To summarize, I will brief you on how to connect all the different parts of this tech post.

- First, get comfortable with the different classifiers namely Naive Bayes, Maximum Entropy and Support Vector Machines. Learn how they work in the background and the math behind it.
- Training data and the features selected for use in the classifier impacts the accuracy of your classifier the most. Look up on the mentioned training data resources already available to train your classifier. I have currently explained how to use unigrams as features, you can include bi-grams, tri-grams and even dictionaries to improve the accuracy of your classifier.
- Once you have have a trained model, extract the tweets for a particular keyword. Clean the tweets and run the classifier on it to extract the labels.
- Build a simple web interface (webpy (http://webpy.org/)) which facilitates the user to enter the keyword and show the result graphically (line chart or column chart using Google Charts (https://developers.google.com/chart/))
- The source code for the "Twitter Sentiment Analyzer" that I built can be found at https://github.com/ravikiranj/twitter-sentiment-analyzer (https://github.com/ravikiranj/twitter-sentiment-analyzer). Good luck building your twitter sentiment classifier :) !
- If you want to tweak and play with Twitter search, check out Twitter REST API Console (https://dev.twitter.com/rest/tools/console). Make sure to check *oAuth* authentication as search API requires authentication.

Related Articles on Web

- Twitter sentiment analysis using Python and NLTK (my inspiration for this tech post) (http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/)
- Sentiment Analysis in Python (http://andybromberg.com/sentiment-analysis-python/)
- An Introduction to Text Mining using Twitter Streaming API and Python (http://adilmoujahid.com/posts/2014/07/twitter-analytics/)
- Sentiment 140 - online tweet analyzer (http://www.sentiment140.com/)
- Text Classification for Sentiment Analysis - Naive Bayes Classifier (http://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/)
- Sanders Analytics - Twitter Sentiment (http://www.sananalytics.com/lab/twitter-sentiment/)
- Natural Language Toolkit (NLTK) (http://www.nltk.org/)

Hey, thanks for Reading! I'm Ravikiran Janardhana (../../../../stories/whoami) and I'm currently building APIs at TripAdvisor (http://www.tripadvisor.com).

Follow @ravikiranj   Tweet 2

Previous post (../../../2011/code/how-implement-infinite-scrolling-using-native-javascript-and-yui3/)

Next post (../../../2013/code/break-my-work-rsi-prevention-software/)

Comments

186 Comments ravikiranj.net Login

Recommend 5 Share Sort by Newest

Join the discussion…

Julie • 7 days ago

Thank you so much for this great article. As I'm writing my thesis based on these concepts, it was and is very helpful. One problem I've encountered: when I train the maxent classifier and try to classify text (into 3 categories: pos, neg and neutral) with that trained classifier, all text gets classified as "neutral". Do you know perhaps how to overcome this? Thank you in advance

Reply • Share ›

ravikiranj Mod → Julie • 6 days ago

How big is your training data ? Are you sure it's not biased towards "neutral" label ? If you have read the theory behind Maximum Entropy a.k.a Multonomial Logistic regression, you can dump the informative features or the explanation of how influential each feature is. Take a look at http://www.nltk.org/_modules/n... documentation. You should explore the following two methods to understand what's going w.r.t features.

```
 def show_most_informative_features(self, n=10, show='all'):
       """
       :param show: all, neg, or pos (for negative-only or positive-only)
       """

def explain(self, featureset, columns=4):
       """
       Print a table showing the effect of each of the features in
       the given feature set, and how they combine to determine the
       probabilities of each label for that featureset.
       """
```

Reply • Share ›

Utsav Jha • 10 days ago

Hello Sir. I read your article and was quite awed by it. However, when i try to run the naive_bayes_classifier_demo.py i get the the error:Name error: set_word_features() isnt defined. I looked at other files as well and thought that this function is a user defined one, whose body hasnt been defined. Please help me out. its quite urgent.

Reply • Share ›

Neha Sood → Utsav Jha • 4 days ago

yes, help please

Reply • Share ›

ravikiranj Mod → Neha Sood • 4 days ago

Please refer to https://github.com/ravikiranj/..., it shows the correct usage of naiveBayes algorithm. I should probably delete the other files which I don't maintain. They were hacky scripts that I wrote 2 years ago and to be frank, I don't even know what they do now.

Reply • Share ›

Utsav Jha → Utsav Jha • 10 days ago

if anybody else has the answer, I would be obliged if you helped me out.

Reply • Share ›

Josh • 25 days ago